Problem 1-1 (Adding Binary Integers) 5 points

You are given two binary integers \( a \) and \( b \), where \( a \) has \( n \) bits, and \( b \) has \( m \) bits, where \( n \geq m \). These integers are stored in two arrays \( A[1\ldots n] \) and \( B[1\ldots m] \), in reverse order. For example, if \( a = 111000 \) and \( b = 1110 \), then \( A[1] = A[2] = A[3] = 0, A[4] = A[5] = A[6] = 1, B[1] = 0, B[2] = B[3] = B[4] = 1 \). Your goal is to produce an array \( C[1\ldots n+1] \) which stores the sum \( c \) of \( a \) and \( b \). For example, \( 111000 + 1110 = 1000110 \), meaning that \( C[1\ldots7] = 0110001 \).

Write the pseudocode to produce \( C \). Why was it convenient to store the arrays “backwards”?

Problem 1-2 (Insertion Sort Variation) 5 points

In class we learned how to implement insertion sort by comparing the key element to the largest element in the sorted portion of the array, and moving that element to the right, if it was larger than the key, and then comparing the key to successively smaller elements until the right position is found.

Implement a variation of insertion sort, in which you instead compare the key to the smallest element in the sorted portion of the array and then iterate by comparing to successively larger elements. How does this algorithm compare in terms of efficiency to the traditional insertion sort?

Problem 1-3 (Asymptotic Comparisons) 10 Points

For each of the following pairs of functions \( f(n) \) and \( g(n) \), state whether \( f \) is \( O(g) \); whether \( f \) is \( o(g) \); whether \( f \) is \( \Theta(g) \); whether \( f \) is \( \Omega(g) \); and whether \( f \) is \( \omega(g) \). (More than one of these can be true for a single pair!)

(a) \( f(n) = 13n^{19} + 92; g(n) = \frac{n^{24} - n^{23} + 5}{5n^{14} + 2000} \).
(b) \( f(n) = \log(n^{19} + 3n); g(n) = \log(n^{0.2} - 1) \).
(c) \( f(n) = \log(5^n + n); g(n) = \log(n^{20}) \).
(d) \( f(n) = n^3 \cdot 3^n; g(n) = n^3 \cdot 4^n \).
(e) \( f(n) = (n^n)^2; g(n) = n^{(n^2)} \).
Problem 1-4 (The Same or Not the Same?)  

The following two functions both take as arguments two \( n \)-element arrays \( A \) and \( B \):

**Magic-1** \((A, B, n)\)

\[
\text{For } i = 1 \text{ to } n \\
\quad \text{For } j = 1 \text{ to } n \\
\quad \quad \text{If } A[i] \geq B[j] \text{ Return FALSE} \\
\text{Return TRUE}
\]

**Magic-2** \((A, B, n)\)

\[
\text{temp} := A[1] \\
\text{For } i = 2 \text{ to } n \\
\quad \quad \text{If } A[i] > \text{ temp} \text{ Then } \text{temp} := A[i] \\
\text{For } j = 1 \text{ to } n \\
\quad \quad \text{If } \text{temp} \geq B[j] \text{ Return FALSE} \\
\text{Return TRUE}
\]

(a) (2 points) It turns out both of these procedures return TRUE if and only if the same ‘special condition’ regarding the arrays \( A \) and \( B \) holds. Describe this ‘special condition’ in English.

(b) (5 points) Analyze the worst-case running time for both algorithms in the \( \Theta \)-notation. Which algorithm would you chose? Is it the one with the shortest code (number of lines)?

(c) (3 points) Does the situation change if we consider the best-case running time for both algorithms?